Distributed
FREE
to over 4,000 Pick installations!

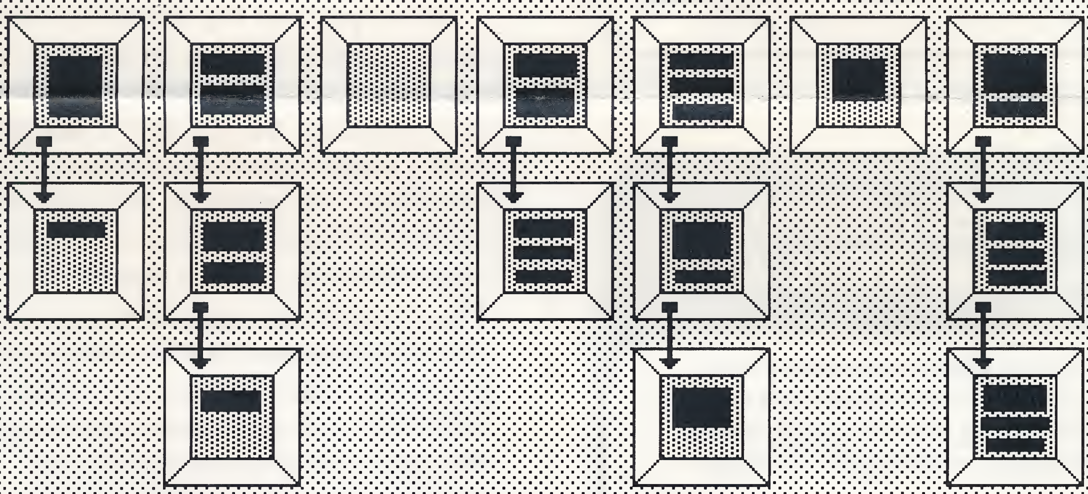
PRAGMA'S

PRODUCT PROFILES

News and Information for Pick™ Operating System Users

Issue Number 22

January 1986



How files grow

207 GRANADA DRIVE
APTOS, CA 95003

BULK RATE
U.S. POSTAGE
PAID
APTOS, CA 95003
Permit No. 67

Address Correction Requested

Pragma's

PRODUCT PROFILES

Issue #22 • January 1986

Product Profiles is published periodically by:

Semaphore Corporation
207 Granada Drive
Aptos, CA 95003
Telephone 408-688-9200

Does Your Mailing Label Say RENEW?

If the mailing label on this issue says RENEW, your free subscription has expired. To avoid missing the next issue, renew immediately by returning this coupon!

Name: ______________________________

Company: ______________________________

Address: ______________________________

City/State/ZIP: ______________________________

Telephone: ______________________________

I use the Pick operating system and my mailing address is in the USA. Please continue sending me *Pragma's Product Profiles* free of charge.

Signature: ______________________________

Date: ______________________________

Return this form to:
Pragma • 207 Granada Drive • Aptos, CA 95003

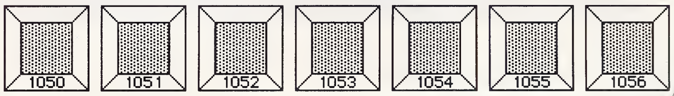
How Files Grow

On computers that use the Pick operating system, all of the available storage space on a disk is divided into chunks called *frames.* In most Pick implementations, a frame can hold 500 characters of data. If a machine has a ten megabyte disk (which means the disk can hold ten million characters of data), then the disk will be divided into 10,000,000/500 or 20,000 frames.

(In actual practice, frames also consume a few characters for identifying each frame and its position on the disk, and disks themselves also allocate part of their available space for such "format" information, so the calculation just given is really only a rough approximation. Use the POVF verb on your computer to determine exactly how many frames are unused and currently available at any given time.)

When you use the CREATE-FILE verb to create a new empty file on the disk, you have to specify a special number called the *modulo,* which indicates the number of frames for storing data that should initially be reserved for the file. For example, the command CREATE-FILE PAYROLL 1 7 creates a file called PAYROLL with a modulo of seven. (The 1 is another required modulo for the *dictionary* portion of the file, which can be ignored for now. The actual syntax of the CREATE-FILE command may be slightly different on your brand of computer, and is described in your owner's manual.)

If we create PAYROLL with a modulo of 7, then the operating system finds seven unused contiguous frames on disk and reserves them for any data we might eventually place in the file. We can think of the PAYROLL file as seven empty picture *frames,* ready to store data:



Notice that each frame is identified with a unique number, called the *frame ID,* which indicates the frame's actual position among

EMULOR
EXTRA TECHNOLOGY

EMULOR COMPUTER SYSTEMS

20 MB System
Suggested List
$2,995.00
Computer Desk
$159.00

RUNS ALL IBM PC*XT COMPATIBLE SOFTWARE!

SYSTEM INCLUDES

- 20 MB Hard Disc Drive
- One 360K Disc Drive
- 512K On Board Memory
- B&W/Color/Graphic Card
- 12" Amber Monitor
- Keyboard
- 9 Expansion Slots
- Serial & Parallel Ports
- Clock Calendar
- 130 Watt Power Supply
- Game & Light Pen Port
- Memory Expandable

Tested and approved by

PICK T.M.
SYSTEMS

WANTED:
VADS
VARS
VAMS
OEMS

FOR MORE INFORMATION

CALL TOLL FREE

* IBM is the trademark of International Business Machines, Inc.

EMULOR
CORPORATION

Outside Calif.: 1-800-233-6330
In Calif.: 1-800-231-0744

2701 Toledo Street Bldg. 706 Torrance, CA 90503

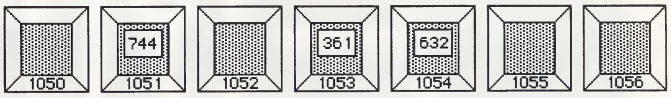
the many thousands of frames on the disk.

Eventually, we'll want to store some *items* in our PAYROLL file. Items are arbitrarily long strings of data that can be created with the computer's Editor or by a BASIC program. For example, items in our PAYROLL file will probably consist of employee names, addresses, salary amounts, and other data.

Just like frames, every item is identified with a unique *item ID,* which is a way to tell items apart from one another when they are all stored in the same file. Item IDs are usually related to the data in the item. For example, payroll files usually use employee numbers for item IDs, while inventory files might use part numbers for item IDs.

```
EMP #361
JOHN SMITH
ANYTOWN USA
845-49-4416
3200/MONTH
```

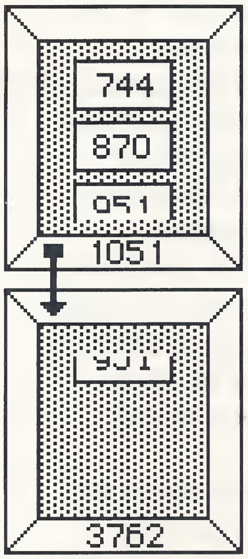
If we use the Editor to create three data items for employees 361, 632 and 744, and save those items in our PAYROLL file, the operating system will save the three items in three different frames of the file:



We can see that frame 1051 contains item 744, frame 1053 contains item 361, and frame 1054 contains items 632. Whenever we *write* an item into a file, or *read* an item from a file, the operating system uses a procedure called *hashing* to automatically determine the frame in which the item will be located. Hashing is the way the operating system tries to evenly spread out items among all frames in the file, so that all of the frames will more or less fill up at the same rate. (The way hashing scatters items in a file explains the apparent random order in which items are shown when you use the LIST verb, which is a command that simply lists item IDs in the order in which they are stored in the frames.)

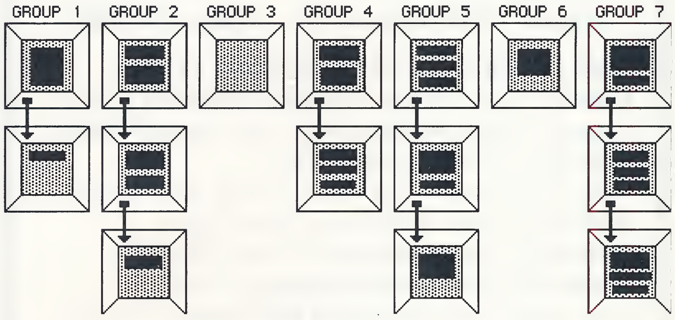
There are many different kinds of hashing techniques, or *algorithms*. The hashing algorithm on our example computer works by simply adding up the digits in the item ID and then dividing by the file's modulo. The remainder of the division is added to the first frame number in the file, giving the frame number where the item should be stored. For example, item ID 744 gives 7+4+4 = 15, and 15 divided by modulo 7 gives a remainder of 1, so item 744 "hashes to" frame 1050+1, or 1051. A mathematician calls that kind of operation *modulo arithmetic,* which is where the term modulo comes from.

Eventually, we can save enough items in the PAYROLL file to cause one of the frames to completely fill up. For example, suppose frame 1051 contains items 744 and 870, and both of the items consist of 200 characters of data, leaving only 100 characters free in the frame. If we create another item 200 characters long with an item ID of 951, the hashing algorithm will try to also save that item in frame 1051. Since there's not enough room left in frame 1051 for another item that large, what happens? The operating system automatically finds another available frame and "links" it to frame 1051, effectively



extending the length of the frame so that the data in it can *overflow.* (While linked frames can actually reside far apart from one another on the disk, the linking process causes the operating system to automatically record the relation between the two frames so that it can remember what frame to jump to whenever it has to extend frames.) As a result of linking frames, the beginning of the new item is stored in the first frame, and the rest of the item's data is stored in the linked frame.

A starting frame and all frames linked to it are together called a *group*. Groups always grow by linking up frames from the pool of unused and available frames listed by the POVF verb (a name that now makes sense if you think of POVF as "print overflow"). The number of groups in a file is always the same as the file's modulo, but each group can end up with a different number of frames as more and more items are saved and the file grows in size:



In conjunction with the size of items, the modulo can have a major impact on how efficiently frame space is used. If the modulo

is too small, too many items will hash to the same group, and groups will overflow frequently and require many linked frames. In that case, space is used efficiently, but much time is wasted after the operating system determines which group an item hashes to, since the operating system must then sequentially search through every item and frame in the group to find one particular piece of data. If the modulo is too large, overflow is eliminated, but many frames will be all or partially empty, and a lot of space will be wasted.

There are many different techniques for trying to determine the perfect choice of modulo for a given file. A good rule of thumb is to make the modulo equal to the total number of characters in a file after dividing by 500, or equal to the number of items in the file, whichever is smaller. And avoid modulos that are multiples of 2 or 5, since they cause poor hashing results for decimal (base 10) hashing algorithms. Once a ballpark modulo is chosen, try the HASH-TEST verb and experiment with other nearby modulo values to find one that achieves a good balance between the total number of frames occupied and the average number of items per group. Δ

Club notices

We've recently received newsletters and notices from the following Pick user groups:

California Data Base Mgmt. Assoc.
Box 3926
Tustin, CA 92681
(213) 559-3039

Colorado Pick Users
Box 3154
Denver, CO 80201
321-8569

National Capital Area Pick Users
2131 Espey Ct. #16
Crofton, MD 21114
261-6305

New England Reality Users
Box 339
Cambridge, MA 02141
(617) 661-0500

Northern California Pick Users
Box 884474
San Francisco, CA 94188
863-5999

Pick Users Group of Arizona
2122 E. Highland Ave. #420
Phoenix, AZ 85016

Pick Users Management Association
144 W. Eagle Rd.
Havertown, PA 19083

Pick Users of Texas
2103 Whirlaway
Stafford, TX 77477

Richmond Area Pick User Group
Box 3409
Richmond, VA 23235
320-7925

Sierra Pick Information Exchange
6403 Coyle Ave. #150
Carmichael, CA 95608

Southern Arizona Pick Users Group
3130 N. Dodge Blvd.
Tucson, AZ 85733

Utah Pick Users Group
207 MBH
Salt Lake City, UT 84112

If your group isn't listed above, call Pragma at (408) 688-9200.

The 68020-based Altos 3068.

If you're a PICK software developer or dealer, you owe it to yourself to see a demonstration of the forty-user Altos 3068.
Call this number, and we'll make the arrangements.
(800) ALTOS U.S.

ALTOS®
COMPUTER SYSTEMS
2641 Orchard Parkway
San Jose, CA 95134

WE SUPPORT PICK.